Chapitre 6 : AWK et SED

Ce sont des filtres programmables

6.1 – AWK

AWK permet d'écrire des traitements numériques sur des fichiers CSV

Alfred **A**ho : compilateurs, algorithmique

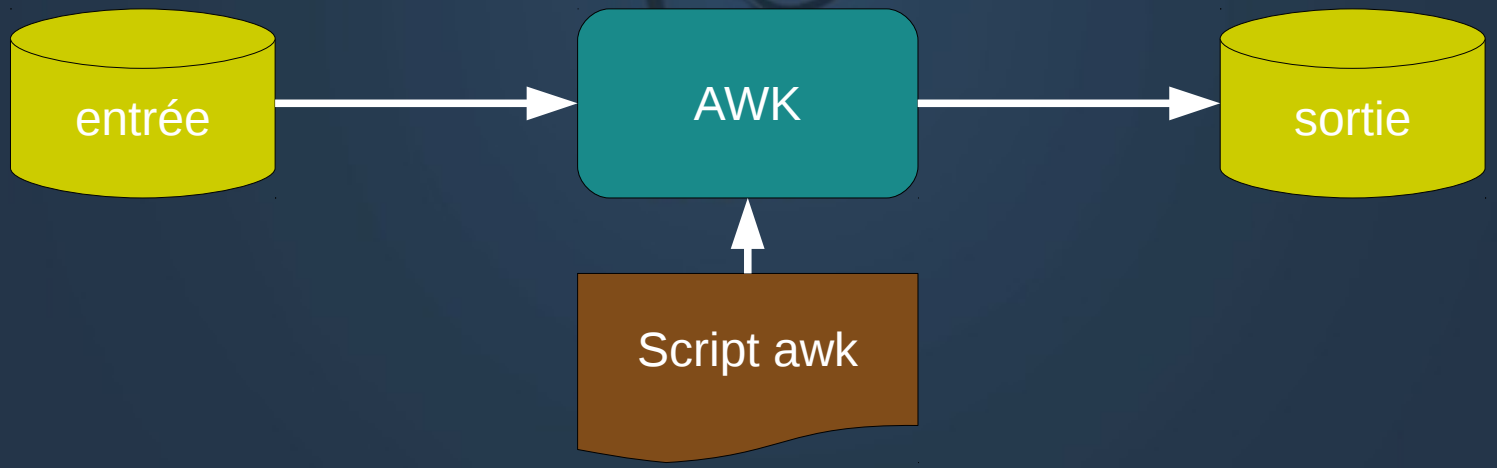
Peter **W**einberger : expressions régulières

Brian **K**ernighan : langage C avec Dennis Ritchie

(Laboratoires Bell 1977)

Principe général de AWK

- 3 éléments :
  - Un texte en entrée (redirection ou tube précédent)
  - Un texte en sortie (redirection ou tube suivant)
  - Un programme appelé script fourni en paramètre



Algo interne de AWK

- Pour chaque ligne du texte d'entrée, AWK exécute les instructions du script
  - La boucle de lecture du fichier d'entrée est implicite
  - On programme seulement le traitement à faire sur chaque ligne
- Les instructions :
  - Affichages
  - Calculs, avec affectation de variables
  - Tests

Exemples

- Affichage du nom et de la taille des fichiers :

```
ls -l | awk '{ print $9 " occupe " $5 " octets" }'
```

- Comptage des fichiers et dossiers

```
ls | awk 'END { print "Il y a " NR " éléments"}'
```

- Affichage de certaines lignes

```
ls -l | awk '($5 > 99999){ print $9 " est grand" }'
```

Mode d'emploi

- Syntaxe de la commande :
  - `awk 'script...'  < entrée`
  - `commandes...| awk 'script' | suite...`
- Autre possibilité, si le script est plus long :
  - saisir le script dans un fichier.awk
  - `awk -f nomscript.awk  < entrée`

Contenu d'un script

- Un script AWK est composé de groupes `condition { instructions }`
  - La condition peut être absente, dans ce cas c'est comme si elle était toujours vraie
  - C'est comme s'il y avait un `if` juste avant, mais il ne faut pas l'écrire
- AWK exécute les instructions sur toutes les lignes des données pour lesquelles la condition est vraie

Conditions

- Les conditions peuvent être :
  - Un test, comme en langage C

```
($5 > 4096) { print $9 }
(NR > 1) { somme = somme + $5 }
```

  - Un motif style egrep, écrit entre deux slash /.../

```
/^d/ { print $0 }
/,FRA,/ { nbF = nbF + 1 }
```

  - La négation d'un motif

```
!/^d/ { print $0 }
```

Conditions spéciales

- Pour initialiser un script, on emploie la condition spéciale BEGIN

```
BEGIN { nombre=0 }
```

- Pour terminer un script, on emploie la condition spéciale END

```
END { print "nombre=" nombre }
```

- Nb : BEGIN et END sont des sortes de booléens, pas des mots clés qui voudraient dire début et fin du script

Instructions

- On les place entre { }, après la condition
- On les sépare par des ;

```
/^d/{ nbDirs = nbDirs+1 ; nbTot = nbTot+1 }
```

- Affichage : print "chaîne" variable ...

```
END { print "il y a " nbF " coureurs français" }
```

- Affectation : variable = valeur
  - Calculs numériques
  - Chaînes de caractères entre "..."

Exemple

- Afficher le nom du plus grand fichier affiché par ls -l, la taille se trouve dans le 5e champ et le nom dans le 9e :

```
ls -l | awk '
   BEGIN { max=0 ; nom="aucun" }
   ($5 > max) { max=$5 ; nom=$9 }
   END { print nom " : " max }'
```

- NB : on peut tout mettre sur la même ligne

Variables prédéfinies

- Il existe des variables prédéfinies, dont :
  - NR numéro de la ligne courante
  - $0 la ligne courante en entier
  - FS séparateur de champ (par défaut c'est espace)
    - En général, on le définit avec la clause BEGIN
  - $1, $2... champs successifs de la ligne
  - NF nombre de champs dans la ligne courante

Opérateurs et fonctions de calcul

- Les opérateurs et fonctions mathématiques sont ceux du langage C : + - * / % ...
- Il y a la trigonométrie, les exponentielles...
  - Exemple :

```
ls -l | awk '{print $9 ":" int(log($5)/log(1024)) }'
```

Ça affiche 0 pour les fichiers plus petits que 1Ko, 1 pour les fichiers dont la taille s'énonce en Ko, 2 pour ceux dont la taille s'énonce en Mo, 3 pour Go

car $log(X)/log(1024) = n$ tel que $1024^n = X$

Fonctions sur les chaînes

- deux chaînes juxtaposées sont concaténées
- match(chaîne, motif) renvoie la position du motif dans la chaîne (1 à ...) ou 0 s'il est absent
- sub(rech,remp,chaîne) remplace rech par remp
- length(chaîne) longueur de la chaîne
- substr(chaîne,déb,fin) extrait un morceau de la chaîne délimité par les indices
- NB : il y a beaucoup d'autres fonctions

Tableaux en AWK

- On peut créer un tableau très facilement :
  - `tableau[indice] = valeur`
  - L'indice peut être un nombre ou une chaîne
- Savoir si un indice est dans un tableau :
  - `if (indice in tableau) {` instructions... `}`
  - `if (tableau[indice] != "") {` instructions... `}`
- Parcourir toutes les valeurs du tableau :
  - `for (indice in tableau) {` instructions... `}`

Exemple

- On veut compter les coureurs de chaque nationalité (champ n°4) dans ce fichier :

```
800,8,Randy DE PUNIET,FRA,Honda,81
125,5,Sandro CORTESE,GER,Derbi,118
250,29,Hector FAUBEL,SPA,Suter,8
250,3,Andrea IANNONE,ITA,Speed Up,144
125,16,Alexis MASBOU,FRA,Aprilia,20
125,7,Esteve RABAT,SPA,Aprilia,108
125,3,Marc MARQUEZ,SPA,Derbi,197
250,16,Fonsi NIETO,SPA,Moriwaki,41
...
```

Le script de l'exemple

- Idée : un tableau Nombre[nomdupays] = nombre de coureurs, avec nomdupays = champ n°4

```
BEGIN { FS="," }
{ Nombre[$4] += 1 }
END {
    for (pays in Nombre) {
        print pays " : " Nombre[pays]
    }
}
```

6.2 – SED

SED permet d'appliquer automatiquement des commandes d'édition sur un fichier

Lee McMahon (Laboratoires Bell 1973)

Vous connaissez déjà un peu SED

- ED est un sous-ensemble de VI
- Dans vi : `:%s/titi/toto/g`

  C'est une commande ED de remplacement d'un mot par un autre
- Dans vi : `:w tutu.txt`

  C'est une commande ED d'enregistrement du texte dans un fichier
- SED = ED en mode « tube », pas interactif

Stream EDitor

- SED est un éditeur d'un texte arrivant sous forme d'un flux (*stream*) = tube venant d'une précédente commande ou redirection d'entrée

  ```
  ... | sed -re 'commandes d'édition' | ...
  ```

- Par exemple :

  ```
  ls -l | sed -re '/^-/ifichier :'
  ```

- Les commandes d'édition sont quasiment celles de vi

Autre emploi de SED

- SED peut aussi fonctionner comme un éditeur de fichiers, mais pas interactif
  - C'est à dire qu'on lui fournit des commandes d'édition à l'avance, mais sans pouvoir s'adapter au fichier, ni voir le résultat au fur et à mesure

    `sed -i -re '`*commandes d'édition*`' fichier`
  - Option -i = *in-place* (modifier le fichier fourni)
  - Option -r = utiliser les jokers étendus de egrep
  - Option -e = voici le script à exécuter sur le fichier

Commande SED à connaître

- Ce qui sert le plus souvent : les remplacements
  - Ex : remplacer automatiquement une valeur par une autre dans un fichier de configuration
- La syntaxe de la commande est :

  `/motif1/s/motif2/remplacement/`
  - Motif1 = quelles lignes ? Jokers = ceux de egrep
  - Motif2 = quel mot à y chercher ? (Jokers = egrep)
  - Remplacement = quoi mettre à la place ? (idem)

Exemple

- Soit un fichier appelé profiles.ini (firefox)

```
[General]
StartWithLastProfile=0

[Profile0]
Name=default
IsRelative=1
Path=oijpoezf7.profile
```

- Remplacer =1 par =0 dans la ligne IsRelative

```
sed -i -re '/^IsRelative=/s/=1/=0/' profiles.ini
```

- Utilité : le faire sur un grand nombre de fichiers

Remplacements

- Dans le mot à remplacer, on peut mentionner une (partie du) motif de sélection à l'aide des références \n° (1 à …) (NB : option -r nécessaire)
- Exemple :
  - Intervertir variable=valeur dans le fichier profile.ini

    ```
    sed -i -re '/=/s/([A-Za-z]+)=(0|1)/\2=>\1/' profiles.ini
    ```

    à comprendre ainsi : dans les lignes où il a un =, remplacer la suite *mot*=0 ou 1 par ce 0 ou 1 suivi de => suivi du mot.

Options de remplacement

- Dans la commande

  `/motif1/s/motif2/remplacement/options`
- On peut ajouter des options :
  - g pour remplacer toutes les occurrences
  - un numéro pour indiquer quelle occurrence
  - I pour ignorer la casse (majuscules = minuscules)
- NB : ce sont les mêmes options dans vi

Et si le / est dans un motif ?

- Exemple : remplacer / par \ dans des chemins

```
sed -ire 's///\/' fichier
```
aïe !

- Il faut doubler le \ car c'est un joker : \\
- On doit utiliser un autre séparateur, en fait le séparateur est le caractère qui suit le s

```
sed -ire 's|/|\\|' fichier
sed -ire 's:/:\\:' fichier
sed -ire 's@/usr/local/@/usr/@'
```

Autre commande utile

- On peut également vouloir supprimer certaines lignes
- La commande est :

```
/motif/d
```

  - motif = quelles lignes sont concernées ?
- Exemple : supprimer la ligne Name=...

```
sed -i -re '/^Name=/d' profiles.ini
```

Dernière commande utile

- On peut également vouloir ajouter du texte dans un fichier
- La commande est :

```
/motif/atexte à ajouter
```

  - motif = après quelle ligne faut-il rajouter ce texte
- Exemple : ajouter une ligne active=false

```
sed -i -re '/^\[Profile0\]/aactive=false'
profiles.ini
```

Structure des commandes SED

- On vient de voir trois commandes :

```
/motif/s/motif2/remplacement/
/motif/d
/motif/atexte à ajouter
```

- Le motif /.../ avant la commande est appelé adresse : il désigne les lignes concernées
- À la place, on peut mettre un numéro ou un couple de numéros n1,n2 qui signifie un intervalle, $ représente la dernière ligne du texte

Numéros de ligne

- Exemple, supprimer les 3 premières lignes du fichier :

```
sed -i -re '1,3d' profiles.ini
```

- Exemple, remplacer les = par : dans toutes les lignes à partir de la 5e

```
sed -i -re '5,$s/=/:/' profiles.ini
```

- NB : c'est pareil avec vi

Voila, c'est tout ce qu'il faut connaître. Le reste est dans la doc et sur stackoverflow.com